Fundamentos de Big Data

Lección 5: Kaggle y los retos de Data Science

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Lección 5. – Kaggle y los retos de Data Science

PRESENTACIÓN Y OBJETIVOS

Llegados a este punto, ya disponemos de los conocimientos necesarios acerca de gráficos. Si bien es cierto que deberemos aprender más cosas en un futuro.

Lo visto hasta el momento pudiera ser suficiente para esta asignatura, pero se ha tratado de condensar en el tema 4 muchas cosas, y añadir en este quinto tema una predicción para el Titanic Dataset.

De esta forma, nos sirve como una nueva introducción a Machine Learning, aunque para ello habrá asignaturas específicas, y por otra parte nos viene bien dado que en la próxima asignatura (Programación Python para Big Data) puede que hagamos algo relacionado con Machine Learning.

Objetivos

- Conocer el Titanic Dataset un poco mejor
- Conocer algunas cosas más de Machine Learning y Kaggle

1. Feature Engineering [1]: Introducción

Feature Engineering

En esta parte podemos hacer uso de la información obtenida y conclusiones.

Para hacerlo lo más simple posible, lo que haremos será elegir solamente algunas columnas.

```
In [57]: df.head()
```

Out[57]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figura 1.1: Feature Engineering (parte 1)

```
In [58]: df.isnull().sum()
```

```
Out[58]: Survived      0
         Pclass        0
         Name          0
         Sex           0
         Age         177
         SibSp         0
         Parch         0
         Ticket        0
         Fare          0
         Cabin       687
         Embarked      2
         dtype: int64
```

Figura 1.2: Feature Engineering (parte 2)

-1- Name - no la tendremos en cuenta por simplificar

df["Name"] = df["Name"].str.extract('([A-Za-z]+).',expand=False)

sería una posible forma de analizar la columna Name, pero no lo haremos.

-2- Age - Usamos el valor promedio de la columna para rellenar los valores faltantes

```
In [59]: df.Age.isnull().sum()
Out[59]: 177

In [60]: df.Age = df.Age.fillna(df.Age.mean())

In [61]: df.Age.isnull().sum()
Out[61]: 0
```

Figura 1.3: Feature Engineering (parte 3)

-3- Ticket - No la tendremos en cuenta por simplificar

```
In [62]: df.Ticket.value_counts()
Out[62]: 1601              7
         347082            7
         CA. 2343          7
         3101295           6
         347088            6
                          ..
         345767            1
         PC 17597          1
         364499            1
         SC/AH Basle 541   1
         111427            1
         Name: Ticket, Length: 681, dtype: int64
```

-4- Cabin - no la tendremos en cuenta por falta de información

```
In [63]: df.Cabin.isnull().sum(), len(df)
Out[63]: (687, 891)
```

Figura 1.4: Feature Engineering (parte 4)

-5- Embarked

```
In [64]: df.Embarked.isnull().sum()
Out[64]: 2

In [65]: df.Embarked.value_counts()
Out[65]: S    644
         C    168
         Q     77
         Name: Embarked, dtype: int64

In [66]: df['Embarked'] = df['Embarked'].fillna('S')
         df.Embarked.value_counts()
Out[66]: S    646
         C    168
         Q     77
         Name: Embarked, dtype: int64
```

Figura 1.5: Feature Engineering (parte 5)

BORRAMOS del DataFrame las columnas mencionadas

```
In [67]: df.head(2)
```

Out[67]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C

```
In [68]: df = df.drop(["Name", "Ticket", "Cabin"], axis=1)
         df.head(2)
```

Out[68]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C

Figura 1.6: Feature Engineering (parte 6)

2. Feature Engineering [2]: Datos Categóricos

In [69]:
```
# Una vez tenemos nuestro DataFrame(df)

# lo siguiente será trabajar con las columnas que tienen texto.
```

Concepto de **datos categóricos**:

- columnas con strings hombre/mujer por ejemplo
- columnas con strings con 3 opcines por ejemplo
- en el caso de Pclass 3 hace referencia a "tercera clase"
- y 3 no vale, mas que 1, y más en este caso, cuya probabilidad de supervivencia es mas baja

In [70]:
```
# pd.get_dummies?
```

In [71]:
```
# drop_first=True porque queremos evitar multicolinearidad
# de hecho en Sex si decimos que es hombre, ya no es por ende, mujer.
# entonces siendo 1 lógico Hombre, por ejemplo, sería redundante.
df = pd.get_dummies(df, columns=['Sex', 'Pclass', 'Embarked'], drop_first=True)
df.head()
```

Out[71]:

	Survived	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3	Embarked_Q	Embarked_S
0	0	22.0	1	0	7.2500	1	0	1	0	1
1	1	38.0	1	0	71.2833	0	0	0	0	0
2	1	26.0	0	0	7.9250	0	0	1	0	1
3	1	35.0	1	0	53.1000	0	0	0	0	1
4	0	35.0	0	0	8.0500	1	0	1	0	1

Figura 2.1: Feature Engineering (parte 7)

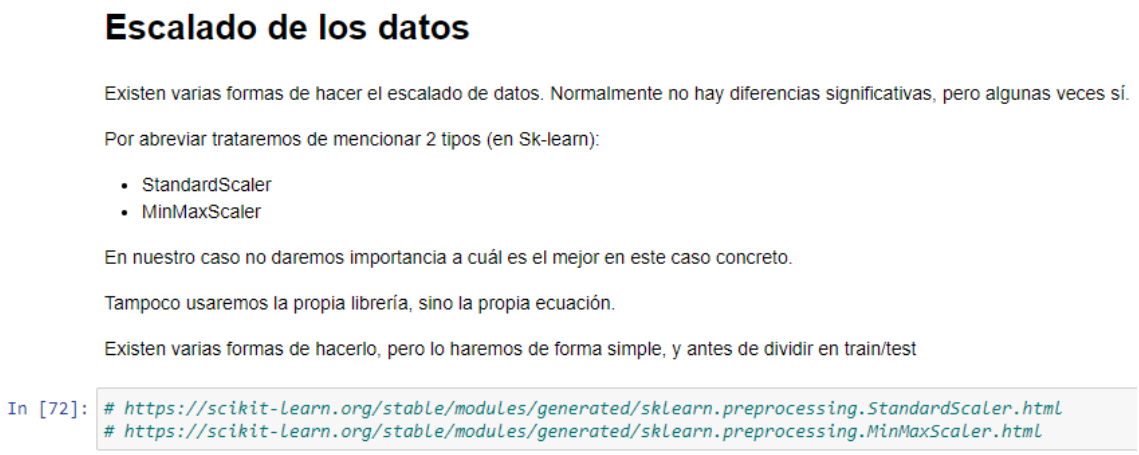
Existen más formas de hacer este tipo de trabajo, pero no podemos profundizar más en este instante, tal vez en un futuro.

3. Feature Engineering [3]: Escalado de los datos

Cuando tenemos diferentes columnas con diferentes escalas, por ejemplo una con valores entre 1 y 10 y otra con valores entre 1 y 1000, no podemos dejarlo así, puesto que a la hora de predecir los algoritmos darían más importancia a aquella con valores más elevados. Tendrían más "peso".

Existen conceptos más técnicos que explican esto, pero no tiene importancia, por el momento.

Lo que hacemos es tratar de escalar los datos, por ejemplo, entre 0 y 1 o entre -1 y +1. Existen varias formas. No es relevante ahora.



Escalado de los datos

Existen varias formas de hacer el escalado de datos. Normalmente no hay diferencias significativas, pero algunas veces sí.

Por abreviar trataremos de mencionar 2 tipos (en Sk-learn):

- StandardScaler
- MinMaxScaler

En nuestro caso no daremos importancia a cuál es el mejor en este caso concreto.

Tampoco usaremos la propia librería, sino la propia ecuación.

Existen varias formas de hacerlo, pero lo haremos de forma simple, y antes de dividir en train/test

In [72]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
```

Figura 3.1: Feature Engineering (parte 8)

In [73]:
```
# StandardScaler

# x-mean(x)/std(x)

df.Age = (df.Age - np.mean(df.Age, axis=0)) / (np.std(df.Age, axis=0))
df.Fare = (df.Fare - np.mean(df.Fare, axis=0)) / (np.std(df.Fare, axis=0))
df.head()
```

Out[73]:

	Survived	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3	Embarked_Q	Embarked_S
0	0	-0.592481	1	0	-0.502445	1	0	1	0	1
1	1	0.638789	1	0	0.786845	0	0	0	0	0
2	1	-0.284663	0	0	-0.488854	0	0	1	0	1
3	1	0.407926	1	0	0.420730	0	0	0	0	1
4	0	0.407926	0	0	-0.486337	1	0	1	0	1

Figura 3.2: Feature Engineering (parte 9)

Por el momento no es necesario entender muy bien todo esto, porque tendrás oportunidad de aprender muchas más cosas en las asignaturas de Machine Learning.

4. Obtención de : X, y

Obtenemos X,y

In [74]:
```
X = df.drop("Survived", axis=1)
X.head()
```

Out[74]:

	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3	Embarked_Q	Embarked_S
0	-0.592481	1	0	-0.502445	1	0	1	0	1
1	0.638789	1	0	0.786845	0	0	0	0	0
2	-0.284663	0	0	-0.488854	0	0	1	0	1
3	0.407926	1	0	0.420730	0	0	0	0	1
4	0.407926	0	0	-0.486337	1	0	1	0	1

In [75]:
```
y = df["Survived"]
y.head()
```

Out[75]:
```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

In [76]:
```
"""X = X.values
y = y.values"""
```

Out[76]:
```
'X = X.values\ny = y.values'
```

Figura 4.1: X,y (parte 1)

Es importante recordar que seleccionamos “X” como “input” y la “y” como “output”.

Existen más formas de hacer eso mismo, pero con hacerlo una vez en este caso es suficiente.

5. TRAIN Y TEST

Entrenamiento y prueba

In [77]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [78]:
```
X_train.head()
```

Out[78]:

	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3	Embarked_Q	Embarked_S
331	1.215947	0	0	-0.074583	1	0	0	0	1
733	-0.515526	0	0	-0.386671	1	1	0	0	1
382	0.177063	0	0	-0.488854	1	0	1	0	1
704	-0.284663	1	0	-0.490280	1	0	1	0	1
813	-1.823750	4	2	-0.018709	0	0	1	0	1

Figura 5.1: Train y Test (parte 1)

In [79]:
```
X_test.head()
```

Out[79]:

	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3	Embarked_Q	Embarked_S
709	0.000000	1	1	-0.341452	1	0	1	0	0
439	0.100109	0	0	-0.437007	1	1	0	0	1
840	-0.746389	0	0	-0.488854	1	0	1	0	1
720	-1.823750	0	1	0.016023	0	1	0	0	1
39	-1.208115	1	0	-0.422074	0	0	1	0	0

In [80]:
```
y_train.head()
```

Out[80]:
```
331    0
733    0
382    0
704    0
813    0
Name: Survived, dtype: int64
```

In [81]:
```
y_test.head()
```

Out[81]:
```
709    1
439    0
840    0
720    1
39     1
Name: Survived, dtype: int64
```

Figura 5.2: Train y Test (parte 2)

6. Algoritmos de Clasificación

Pruebo posibles algoritmos

In [82]:
```
# KNeighborsClassifier
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc_KN = accuracy_score(y_test, y_pred)
acc_KN
```

Out[82]: 0.8212290502793296

In [83]:
```
# DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc_DT = accuracy_score(y_test, y_pred)
acc_DT
```

Out[83]: 0.776536312849162

In [84]:
```
# RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc_RF = accuracy_score(y_test, y_pred)
acc_RF
```

Out[84]: 0.8212290502793296

Figura 6.1: Algoritmos de Clasificación (parte 1)

```
In [85]: # GaussianNB
         clf = GaussianNB()
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         acc_NB = accuracy_score(y_test, y_pred)
         acc_NB

Out[85]: 0.7653631284916201

In [86]: # SVC
         clf = SVC()
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         acc_SVC = accuracy_score(y_test, y_pred)
         acc_SVC

Out[86]: 0.8156424581005587
```

Figura 6.2: Algoritmos de Clasificación (parte 2)

Busco el que a priori mejor predice

```
In [87]: # A priori, y sin ver más parámetros..

         # KNeighborsClassifier
         clf = KNeighborsClassifier()
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         acc_KN = accuracy_score(y_test, y_pred)
         acc_KN

Out[87]: 0.8212290502793296
```

Figura 6.3: Algoritmos de Clasificación (parte 3)

7. ANALIZO LA INFORMACIÓN QUE TENGO EN TEST.CSV

Utilizo ese entrenamiento para test.csv

Para evitar complejidad, y una posible explicación sobre **persistencia del modelo**,

https://scikit-learn.org/stable/modules/model_persistence.html

lo que haremos será hacer esos mismos cambios en test.csv

```
In [88]: test = pd.read_csv("C:/Users/Manut/Desktop/apuntes_big_data_1/TEMA 4/data/test.csv")
         test.head(2)
```

Out[88]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S

Figura 7.1: Test.csv (parte 1)

```
In [89]: df_original = pd.read_csv("C:/Users/Manut/Desktop/apuntes_big_data_1/TEMA 4/data/train.csv")
         df_original.head(2)
```

Out[89]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C

Figura 7.2: Test.csv (parte 2)

LLAMA LA ATENCIÓN ALGO ¿?

Ahora NO TENEMOS la información de "Survived"

TENDREMOS, PUES QUE PREDECIRLA..

Pasos:

- -1- Haremos a cada columna exactamente los mismos cambios en "test" que en "df"
- -2- Haremos la predicción (ahora no sabemos de momento como de buena o mala fue)
- -3- Nos iremos a Kaggle para enviar resultados

Figura 7.3: Test.csv (parte 3)

8. HAGO EN "TEST" LOS CAMBIOS QUE HICE EN "DF"

```
In [90]: test.describe()
```

Out[90]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200

```
In [91]: test.isnull().sum()
```

Out[91]:

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

Figura 8.1: Cambios en Test (parte 1)

```
In [92]: # df.Age = df.Age.fillna(df.Age.mean())
         test.Age = test.Age.fillna(test.Age.mean())
         test.Fare = test.Fare.fillna(test.Fare.mean())

In [93]: test.isnull().sum()

Out[93]: PassengerId      0
         Pclass           0
         Name             0
         Sex              0
         Age              0
         SibSp            0
         Parch            0
         Ticket           0
         Fare             0
         Cabin          327
         Embarked         0
         dtype: int64
```

Figura 8.2: Cambios en Test (parte 2)

```
In [94]: # df = df.drop(["Name", "Ticket", "Cabin"], axis=1)
         test = test.drop(["Name", "Ticket", "Cabin"], axis=1)
         test.head()
```

Out[94]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	male	34.5	0	0	7.8292	Q
1	893	3	female	47.0	1	0	7.0000	S
2	894	2	male	62.0	0	0	9.6875	Q
3	895	3	male	27.0	0	0	8.6625	S
4	896	3	female	22.0	1	1	12.2875	S

Figura 8.3: Cambios en Test (parte 3)

In [95]:
```
# df.Age = (df.Age - np.mean(df.Age, axis=0)) / (np.std(df.Age, axis=0))
# df.Fare = (df.Fare - np.mean(df.Fare, axis=0)) / (np.std(df.Fare, axis=0))

test.Age = (test.Age - np.mean(test.Age, axis=0)) / (np.std(test.Age, axis=0))
test.Fare = (test.Fare - np.mean(test.Fare, axis=0)) / (np.std(test.Fare, axis=0))
test.head()
```

Out[95]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	male	0.334993	0	0	-0.498407	Q
1	893	3	female	1.325530	1	0	-0.513274	S
2	894	2	male	2.514175	0	0	-0.465088	Q
3	895	3	male	-0.259330	0	0	-0.483466	S
4	896	3	female	-0.655545	1	1	-0.418471	S

Figura 8.4: Cambios en Test (parte 4)

In [96]:
```
# df = pd.get_dummies(df, columns=['Sex', 'Pclass', 'Embarked'], drop_first=True)
test = pd.get_dummies(test, columns=['Sex', 'Pclass', 'Embarked'], drop_first=True)
test.head()
```

Out[96]:

	PassengerId	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3	Embarked_Q	Embarked_S
0	892	0.334993	0	0	-0.498407	1	0	1	1	0
1	893	1.325530	1	0	-0.513274	0	0	1	0	1
2	894	2.514175	0	0	-0.465088	1	1	0	1	0
3	895	-0.259330	0	0	-0.483466	1	0	1	0	1
4	896	-0.655545	1	1	-0.418471	0	0	1	0	1

In [97]:
```
# df.head()
```

Figura 8.5: Cambios en Test (parte 5)

In [98]:
```
test = test.drop("PassengerId", axis=1)
test.head()
```

Out[98]:

	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3	Embarked_Q	Embarked_S
0	0.334993	0	0	-0.498407	1	0	1	1	0
1	1.325530	1	0	-0.513274	0	0	1	0	1
2	2.514175	0	0	-0.465088	1	1	0	1	0
3	-0.259330	0	0	-0.483466	1	0	1	0	1
4	-0.655545	1	1	-0.418471	0	0	1	0	1

Figura 8.6: Cambios en Test (parte 6)

9. Predicción usando Test

```
In [99]: # KNeighborsClassifier - el "mejor" bajo estas premisas
         # y en este caso para "test"
         clf = KNeighborsClassifier()
         # entreno con los datos que tenia del primer dataset
         clf.fit(X_train, y_train)
         # ahora hago la predicción sobre "test"
         y_predecida = clf.predict(test)
         y_predecida

Out[99]: array([0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
                1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
                1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
                1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0,
                1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1,
                0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
                0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
                1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
                0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0,
                1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1,
                1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
                0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
                0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
                1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
                1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
                0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0],
               dtype=int64)
```

Figura 9.1: Predicción en Test (parte 1)

En este caso obtenemos esta predicción.

Para tratar de mejorar en el número de aciertos, que después comprobaremos, sería cuestión de mejorar las partes anteriores con nuevos algoritmos, ajustes sobre los mismos, mejor preprocesamiento de datos, etc.

(Supongo será visto en temas de Machine Learning)

10. TRABAJO CON SUBMISSION.CSV

Me creo un dataframe con esa información

In [100]:
```
df_submission = pd.read_csv("C:/Users/Manut/Desktop/apuntes_big_data_1/TEMA 4/data/gender_submission.csv")
df_submission.head()
```

Out[100]:

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

Ahora le pego la información DE MI PREDICCION

In [101]:
```
df_submission["Survived"] = y_predecida
df_submission.head()
```

Out[101]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0

Figura 10.1: Submission (parte 1)

Leemos el dataset ejemplo, y le pegamos nuestro resultado.

A continuación veremos qué hacer con ello.

11. PREDICCIÓN EN KAGGLE DE TITANIC DATASET

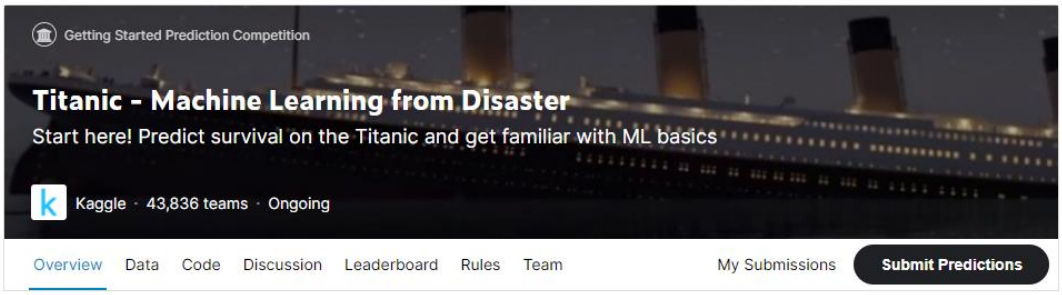


Figura 11.1: Predicción en Kaggle (parte 1)

En "Submit Predictions" podemos enviar nuestros .csv con las predicciones, para que se valoren los aciertos, y el accuracy que obtenemos.

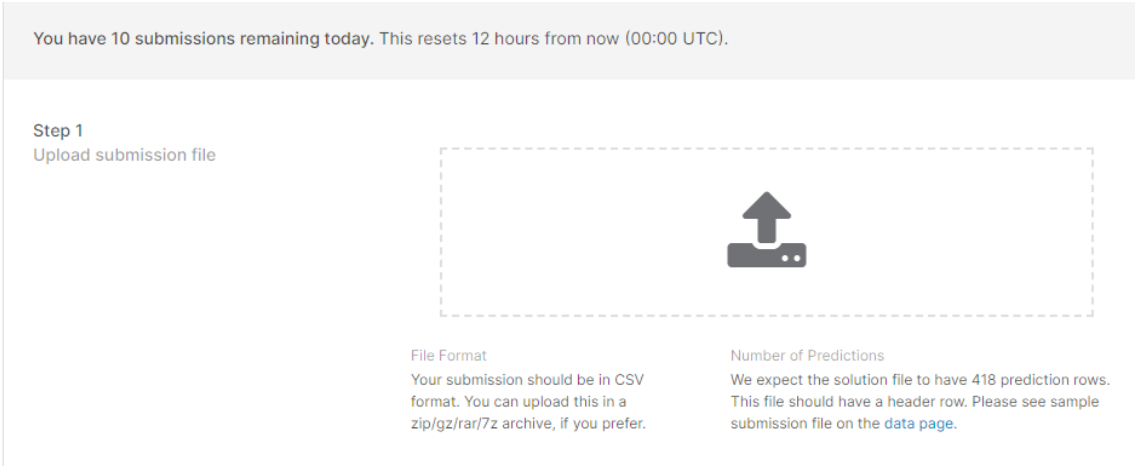


Figura 11.2: Predicción en Kaggle (parte 2)

En la flecha hacemos click y subimos nuestro .CSV.

En mi caso: "titanic.csv"

Figura 11.3: Predicción en Kaggle (parte 3)

Que ya ha subido.

Podemos poner un comentario, y "Make Submission"

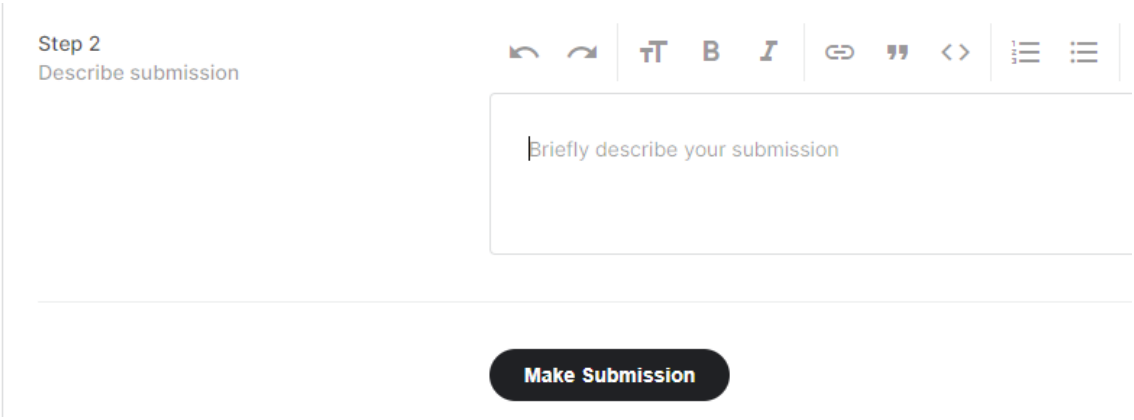


Figura 11.4: Predicción en Kaggle (parte 4)

Lo ideal es irse primero al CSV, en mi caso he podido ver que devuelve un error, entonces.

¿Y cómo soluciono ese error con Microsoft Excel?

12. TRABAJO CON EXCEL INCLUSO SI LO NECESITO

Me voy al .CSV

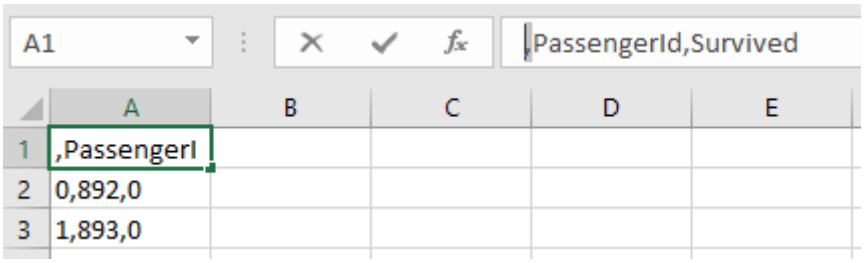


Figura 12.1: Soluciono con Excel el formato (parte 1)

Datos → Texto en columnas

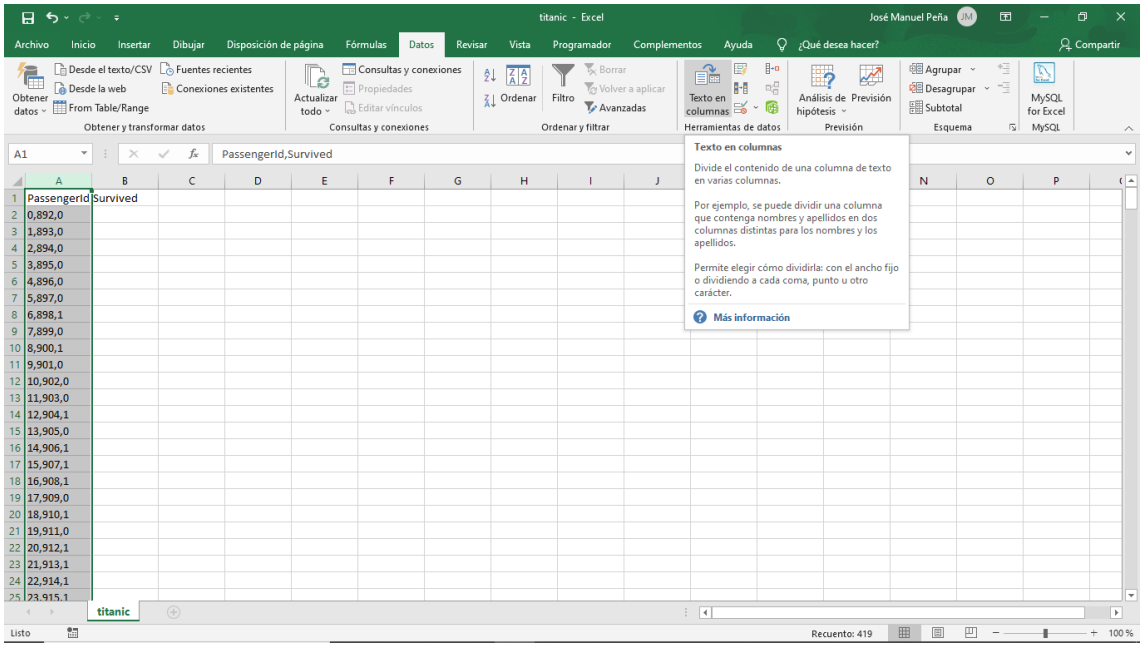


Figura 12.2: Soluciono con Excel el formato (parte 2)

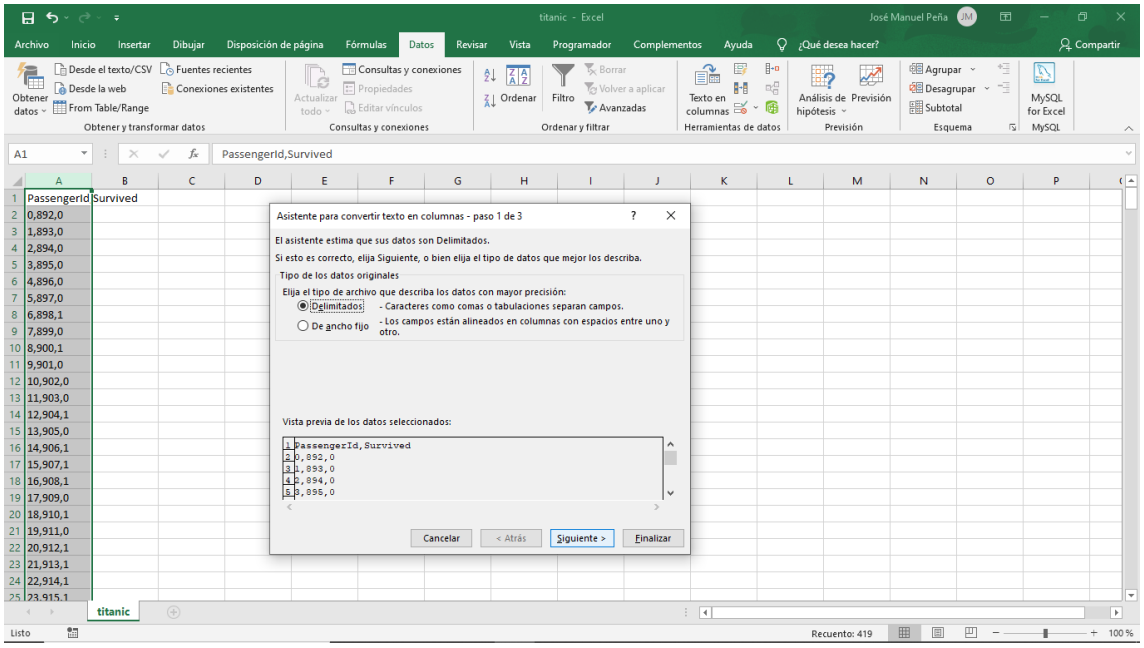


Figura 12.3: Soluciono con Excel el formato (parte 3)

Siguiente

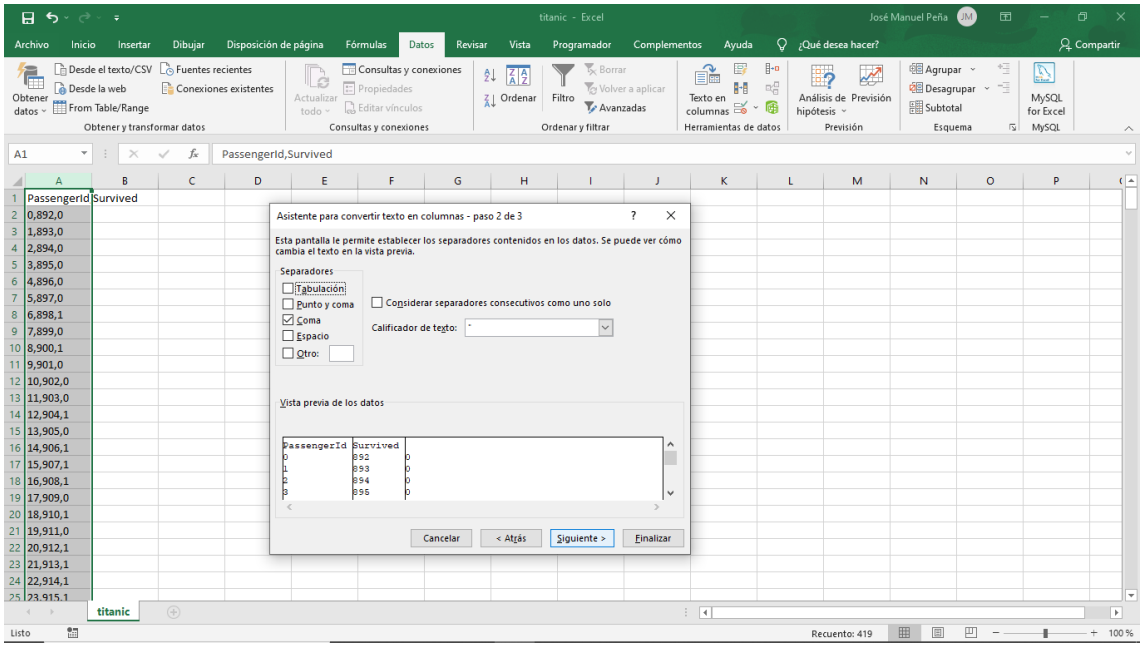


Figura 12.4: Soluciono con Excel el formato (parte 4)

Siguiente, finalizar.

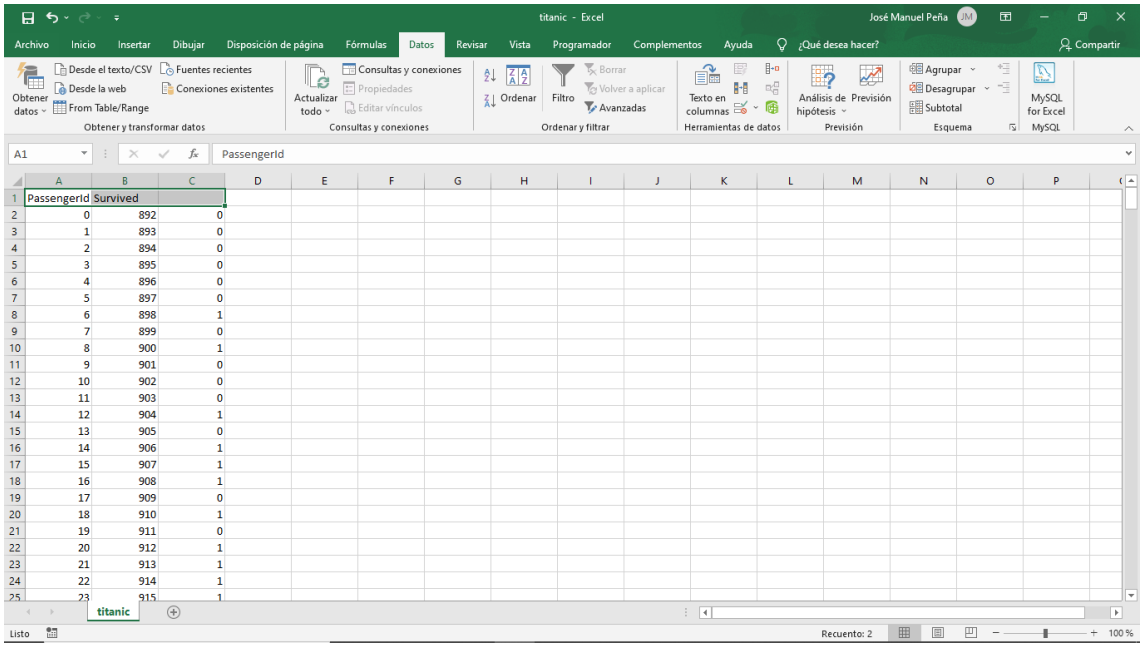


Figura 12.5: Soluciono con Excel el formato (parte 5)

Entonces quito la primera columna y ordeno la info.

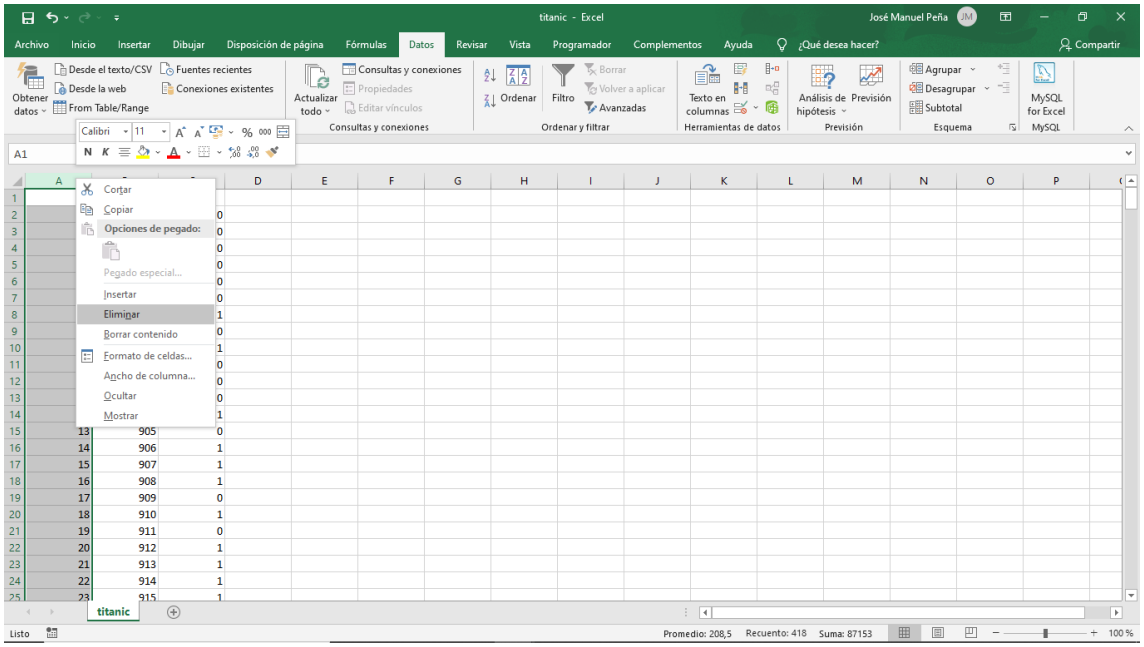


Figura 12.6: Soluciono con Excel el formato (parte 6)

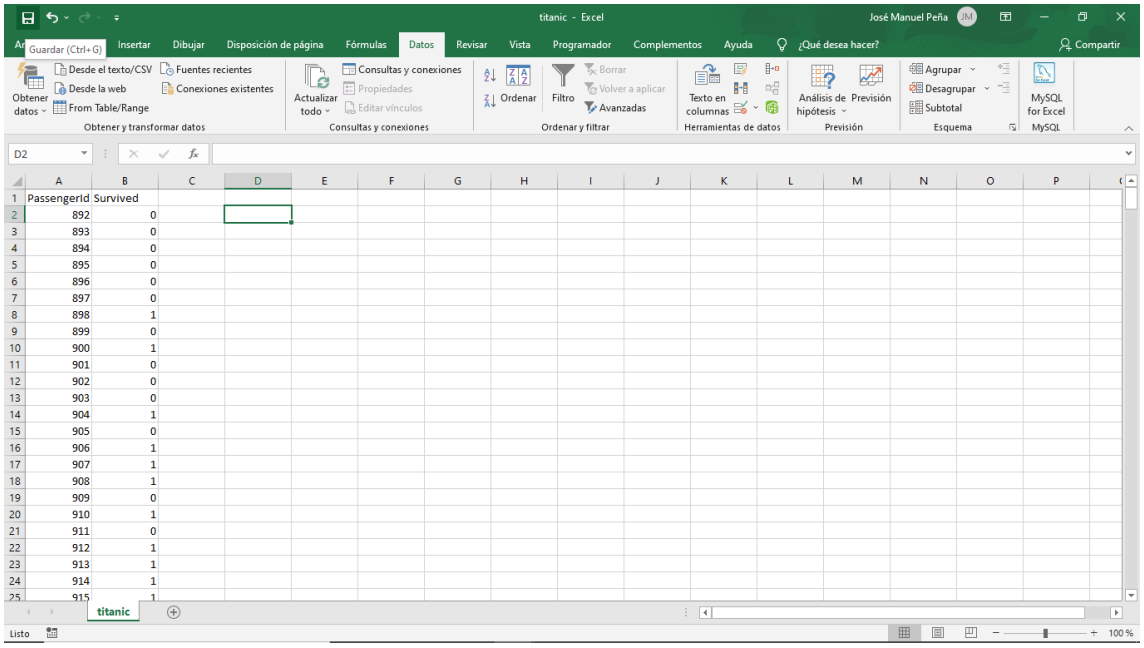


Figura 12.7: Soluciono con Excel el formato (parte 7)

No obstante, lo ideal es hacerlo bien, indicando: “index=False”.

df_submission.to_csv("C:/Users/Manut/Desktop/apuntes_big_data_1/TEMA 4/predicciones/titanic2.csv", **index=False**)

y una vez ejecutamos la opción de “index_false” que yo guarde como: titanic2.csv..

13. EL MOMENTO ESPERADO. ¿QUÉ RESULTADO OBTENGO?

Me devuelve este valor: 0.74, que no es el mejor, pero para ser el primer ejemplo, es suficiente.

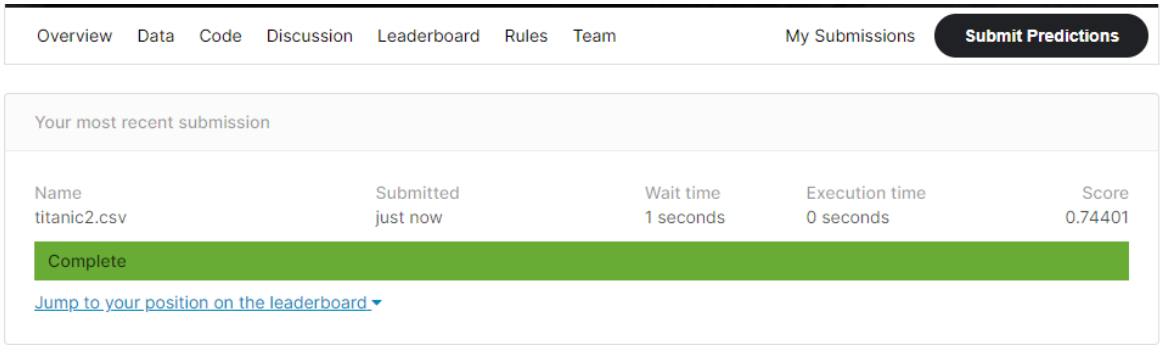


Figura 13.1: Resultado final en Kaggle (parte 1)

Y podemos irnos a la lista general, para valorar en función del resto de participantes en el reto. (Resultado de una de las cuentas de prueba del docente solo para fines formativos en la presente Asignatura, tras realizar una ligera mejora).

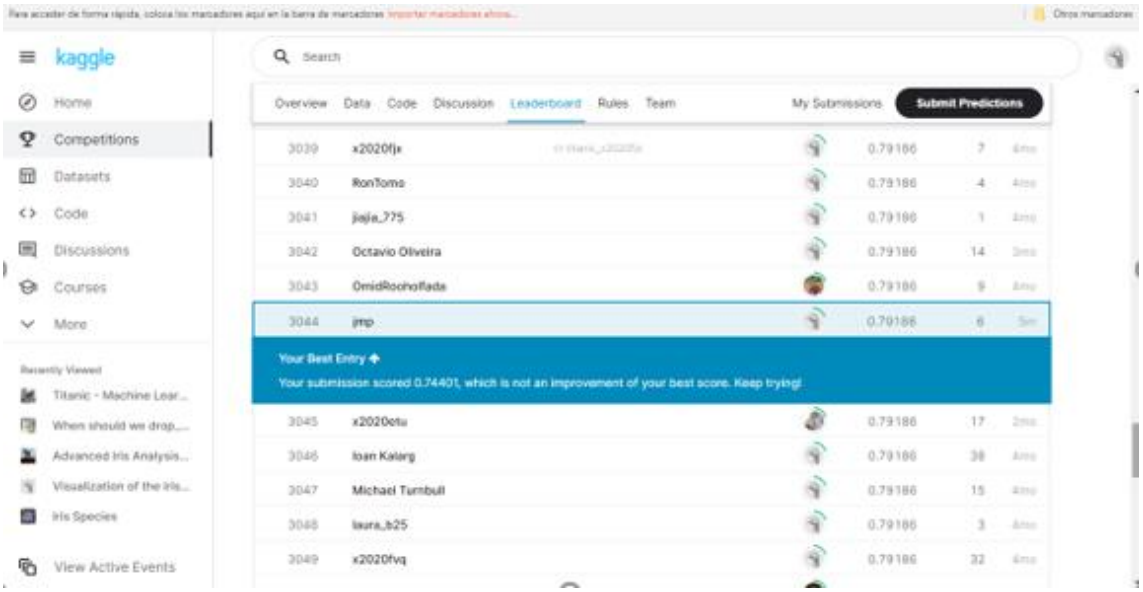


Figura 13.2: Resultado final en Kaggle (parte 2)

14. ¿ PUEDO MEJORAR ESE RESULTADO ?

Sí, en eso consiste.

No te lo muestro, pero yo sí lo hice, entonces tú también puedes hacerlo !

En mi caso siempre trato de aprender las técnicas, que de eso se trata inicialmente, pero lo ideal es llegar por lo menos en cada Dataset hasta el TOP 10% a nivel mundial, o incluso mejor. Pero, es importante conceder importancia al aprendizaje, no solamente a una posición en un ranking.

En este caso hay muchos miles de participantes. Y estamos (o estuvimos) en la posición 3044 a nivel mundial.

A partir de ahora, lo ideal sería esperar a las asignaturas de Machine Learning, aprender cuantas más cosas mejor, y una vez uno/a sienta que está más preparado, regresar a esta competición para tratar de situarse en el TOP 10% a nivel mundial, TOP 7% o incluso TOP 1%.

Todo es cuestión de práctica!

En nuestro caso simplemente hemos explicado un poco después de hacer todas las gráficos porque necesitaremos tener una idea sobre ello en la 2ª Asignatura de Big Data.

Esperamos, por lo menos, haber desperado tu interés en resolver este tipo de retos. Con el tiempo descubrirás que hay grandes oportunidades en esta y otras plataformas para aprender, entre otras cosas.

15. PUNTOS CLAVE

- Existen muchas opciones para aprender Data Science, siendo Kaggle una de ellas.
- El Dataset del Titanic es una buena elección para aprender a hacer Gráficos de muchos tipos.